

SQL (Structured Query Language)

SQL is a database computer language designed for the retrieval and management of data in relational database. SQL stands for Structured Query Language.

SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as standard database language.

Why SQL?

1. Allows users to access data in relational database management systems.
2. Allows users to describe the data.
3. Allows users to define the data in database and manipulate that data.
4. Allows to embed within other languages using SQL modules, libraries & pre-compilers.
5. Allows users to create and drop databases and tables.
6. Allows users to create view, stored procedure, functions in a database.
7. Allows users to set permissions on tables, procedures, and views

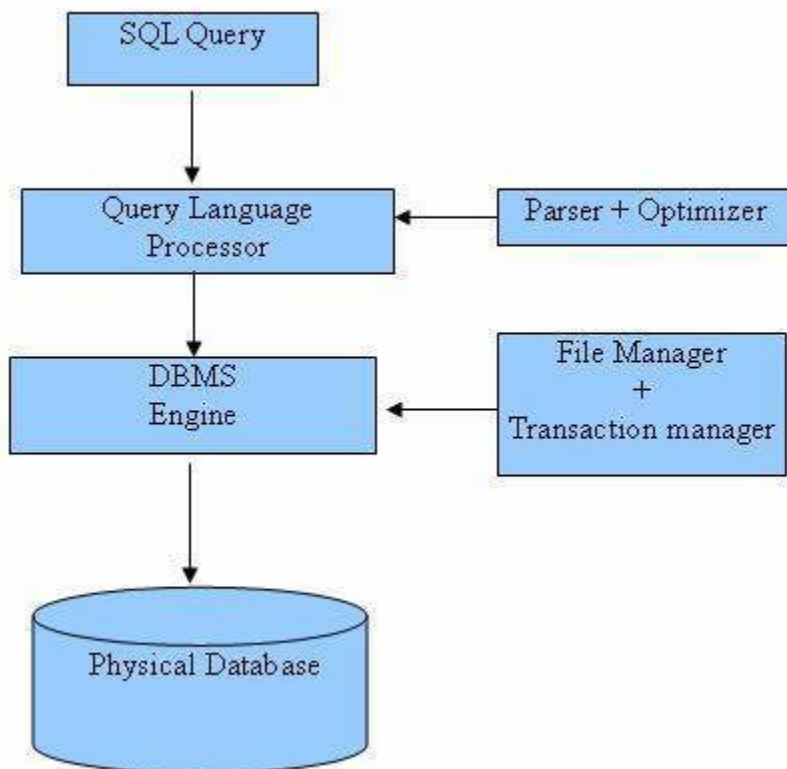
SQL Process:

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

There are various components included in the process. These components are Query Dispatcher, Optimization Engines, Classic Query Engine and SQL Query Engine, etc. Classic query engine handles all non-SQL queries but SQL query engine won't handle logical files.

Following is a simple diagram showing SQL Architecture:

SQL Architecture



SQL Commands:

The standard SQL commands to interact with relational databases are **CREATE**, **SELECT**, **INSERT**, **UPDATE**, **DELETE** and **DROP**.

These commands can be classified into groups based on their nature:

DDL - Data Definition Language:

Command	Description
CREATE	Creates a new table, a view of a table, or other object in database
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other object in the database.

DML - Data Manipulation Language:

Command	Description
SELECT	Retrieves certain records from one or more tables
INSERT	Creates a record
UPDATE	Modifies records
DELETE	Deletes records

DCL - Data Control Language:

Command	Description
GRANT	Gives a privilege to user
REVOKE	Takes back privileges granted from user

RDBMS (Relational Database Management Systems) CONCEPTS

RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

Table

The data in RDBMS is stored in database objects called **tables**. The table is a collection of related data entries and it consists of columns and rows.

Remember, a table is the most common and simplest form of data storage in a relational database. Following is the example of a **CUSTOMERS** table:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Field

Every table is broken up into smaller entities called fields. The fields in the **CUSTOMERS** table consist of **ID**, **NAME**, **AGE**, **ADDRESS** and **SALARY**.

A **field** is a column in a table that is designed to maintain specific information about every record in the table.

Record or row

A record, also called a row of data, is each individual entry that exists in a table.

For example there are 7 records in the above **CUSTOMERS** table.

Following is a single row of data or record in the **CUSTOMERS** table:

1	Ramesh	32	Ahmedabad	2000.00
---	--------	----	-----------	---------

A record is a horizontal entity in a table.

Column

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

For example, a column in the CUSTOMERS table is ADDRESS, which represents location description and would consist of the following:

ADDRESS
Ahmedabad
Delhi
Kota
Mumbai
Bhopal
MP
Indore

NULL value

A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.

It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation.

SQL Constraints

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be column level or table level.

Column level constraints are applied only to one column whereas table level constraints are applied to the whole table.

Following are commonly used constraints available in SQL:

1. **NOT NULL Constraint:** Ensures that a column cannot have NULL value.
2. **DEFAULT Constraint:** Provides a default value for a column when none is specified.
3. **UNIQUE Constraint:** Ensures that all values in a column are different.
4. **PRIMARY Key:** Uniquely identified each rows/records in a database table.
5. **FOREIGN Key:** Uniquely identified a rows/records in any another database table.
6. **CHECK Constraint:** The CHECK constraint ensures that all values in a column satisfy certain conditions.
7. **INDEX:** Use to create and retrieve data from the database very quickly.

Data Integrity:

The following categories of the data integrity exist with each RDBMS:

1. **Entity Integrity:** There are no duplicate rows in a table.
2. **Domain Integrity:** Enforces valid entries for a given column by restricting the type, the format, or the range of values.
3. **Referential integrity:** Rows cannot be deleted, which are used by other records.
4. **User-Defined Integrity:** Enforces some specific business rules that do not fall into entity, domain or referential integrity

Database Normalization

Database normalization is the process of efficiently organizing data in a database.

There are two reasons of the normalization process:

- 1. Eliminating redundant data, for example, storing the same data in more than one tables.**
- 2. Ensuring data dependencies make sense.**

Both of these are worthy goals as they reduce the amount of space a database consumes and ensure that data is logically stored. Normalization consists of a series of guidelines that help guide you in creating a good database structure.

Normalization guidelines are divided into normal forms; think of form as the format or the way a database structure is laid out. The aim of normal forms is to organize the database structure so that it complies with the rules of first normal form, then second normal form, and finally third normal form.

It's your choice to take it further and go to fourth normal form, fifth normal form, and so on, but generally speaking, third normal form is enough.

1. First Normal Form 1NF
2. Second Normal Form 2NF
3. Third Normal Form 3NF

SQL Syntax

SQL is followed by unique set of rules and guidelines called Syntax.

All the SQL statements start with any of the keywords like **SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW** and all the statements end with a semicolon ; .

Important point to be noted is that SQL is **NOT case sensitive**, which means **SELECT** and **select** have same meaning in SQL statements, but MySQL makes difference in table names. So if you are working with MySQL, then you need to give table names as they exist in the database.

SQL SELECT Statement:

```
SELECT column1, column2....columnN  
FROM    table_name;
```

SQL DISTINCT Clause:

```
SELECT DISTINCT column1, column2....columnN  
FROM    table_name;
```

SQL WHERE Clause:

```
SELECT column1, column2....columnN  
FROM    table_name  
WHERE   CONDITION;
```

SQL AND/OR Clause:

```
SELECT column1, column2....columnN  
FROM    table_name  
WHERE   CONDITION-1 {AND|OR} CONDITION-2;
```

SQL IN Clause:

```
SELECT column1, column2....columnN  
FROM    table_name  
WHERE   column_name IN (val-1, val-2, ...val-N);
```

SQL BETWEEN Clause:

```
SELECT column1, column2....columnN  
FROM    table_name  
WHERE   column_name BETWEEN val-1 AND val-2;
```

SQL LIKE Clause:

```
SELECT column1, column2....columnN  
FROM    table_name  
WHERE   column_name LIKE { PATTERN };
```

SQL ORDER BY Clause:

```
SELECT column1, column2....columnN  
FROM   table_name  
WHERE  CONDITION  
ORDER BY column_name {ASC|DESC};
```

SQL GROUP BY Clause:

```
SELECT SUM(column_name)  
FROM   table_name  
WHERE  CONDITION  
GROUP BY column_name;
```

SQL COUNT Clause:

```
SELECT COUNT(column_name)  
FROM   table_name  
WHERE  CONDITION;
```

SQL HAVING Clause:

```
SELECT SUM(column_name)  
FROM   table_name  
WHERE  CONDITION  
GROUP BY column_name  
HAVING (arithmetic function condition);
```

SQL CREATE TABLE Statement:

```
CREATE TABLE table_name(  
column1 datatype,  
column2 datatype,  
column3 datatype,  
.....  
columnN datatype,  
PRIMARY KEY( one or more columns )  
);
```

SQL DROP TABLE Statement:

```
DROP TABLE table_name;
```

SQL CREATE INDEX Statement :

```
CREATE UNIQUE INDEX index_name  
ON table_name ( column1, column2, ...columnN);
```

SQL DROP INDEX Statement :

```
ALTER TABLE table_name  
DROP INDEX index_name;
```

SQL DESC Statement :

```
DESC table_name;
```

SQL TRUNCATE TABLE Statement:

```
TRUNCATE TABLE table_name;
```

SQL ALTER TABLE Statement:

```
ALTER TABLE table_name {ADD|DROP|MODIFY} column_name {data_type};
```

SQL ALTER TABLE Statement *Rename* :

```
ALTER TABLE table_name RENAME TO new_table_name;
```

SQL INSERT INTO Statement:

```
INSERT INTO table_name( column1, column2....columnN)  
VALUES ( value1, value2....valueN);
```

SQL UPDATE Statement:

```
UPDATE table_name  
SET column1 = value1, column2 = value2....columnN=valueN  
[ WHERE CONDITION ];
```

SQL DELETE Statement:

```
DELETE FROM table_name  
WHERE {CONDITION};
```

SQL CREATE DATABASE Statement:

```
CREATE DATABASE database_name;
```

SQL DROP DATABASE Statement:

```
DROP DATABASE database_name;
```

SQL USE Statement:

```
USE database_name;
```

SQL COMMIT Statement:

```
COMMIT;
```

SQL ROLLBACK Statement:

```
ROLLBACK;
```

SQL - SELECT DATABASE, USE STATEMENT

When you have multiple databases in your SQL Schema, then before starting your operation, you would need to select a database where all the operations would be performed.

The SQL USE statement is used to select any existing database in SQL schema.

Syntax:

Basic syntax of USE statement is as follows:

```
USE DatabaseName;
```

Always database name should be unique within the RDBMS.

Example:

You can check available databases as follows:

```
SQL> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| AMROOD |
| TUTORIALSPPOINT |
| mysql |
| orig |
| test |
+-----+
6 rows in set (0.00 sec)
```

Now, if you want to work with AMROOD database, then you can execute the following SQL command and start working with AMROOD database:

```
SQL> USE AMROOD;
```

SQL- CREATE DATABASE

The SQL **CREATE DATABASE** statement is used to create new SQL database.

Syntax:

Basic syntax of CREATE DATABASE statement is as follows:

```
CREATE DATABASE DatabaseName;
```

Always database name should be unique within the RDBMS.

Example:

If you want to create new database <testDB>, then CREATE DATABASE statement would be as follows:

```
SQL> CREATE DATABASE testDB;
```

Make sure you have admin privilege before creating any database. Once a database is created, you can check it in the list of databases as follows:

```
SQL> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| AMROOD |
| TUTORIALSPOINT |
| mysql |
| orig |
| test |
| testDB |
+-----+
7 rows in set (0.00 sec)
```

SQL- Drop or Delete Database

The SQL DROP DATABASE statement is used to drop an existing database in SQL schema.

Syntax:

Basic syntax of DROP DATABASE statement is as follows:

```
DROP DATABASE DatabaseName;
```

Always database name should be unique within the RDBMS.

Example:

If you want to delete an existing database <testDB>, then **DROP DATABASE** statement would be as follows:

```
SQL> DROP DATABASE testDB;
```

NOTE: Be careful before using this operation because by deleting an existing database would result in loss of complete information stored in the database.

Make sure you have admin privilege before dropping any database. Once a database is dropped, you can check it in the list of databases as follows:

```
SQL> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| AMROOD          |
| TUTORIALSPPOINT  |
| mysql           |
| orig            |
| test           |
+-----+
6 rows in set (0.00 sec)
```

SQL- CREATE TABLE

Creating a basic table involves naming the table and defining its columns and each column's data type.

The **SQL CREATE TABLE** statement is used to create a new table.

Syntax:

Basic syntax of CREATE TABLE statement is as follows:

```
CREATE TABLE table_name(  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
    ....  
    columnN datatype,  
    PRIMARY KEY( one or more columns )  
);
```

CREATE TABLE is the keyword telling the database system what you want to do. In this case, you want to create a new table. The unique name or identifier for the table follows the **CREATE TABLE** statement.

Then in brackets comes the list defining each column in the table and what sort of data type it is.

The syntax becomes clearer with an example below.

Example:

Following is an example, which creates a CUSTOMERS table with ID as primary key and NOT NULL are the constraints showing that these fields cannot be NULL while creating records in this table:

```
SQL> CREATE TABLE CUSTOMERS(  
    ID      INT                NOT NULL,  
    NAME    VARCHAR (20)       NOT NULL,  
    AGE     INT                NOT NULL,  
    ADDRESS CHAR (25) ,  
    SALARY  DECIMAL (18, 2),  
    PRIMARY KEY (ID)  
);
```

You can verify if your table has been created successfully by looking at the message displayed by the SQL server, otherwise you can use **DESC** command as follows:

```
SQL> DESC CUSTOMERS;
```

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI		
NAME	varchar(20)	NO			
AGE	int(11)	NO			
ADDRESS	char(25)	YES		NULL	
SALARY	decimal(18,2)	YES		NULL	

5 rows in set (0.00 sec)

Now, you have **CUSTOMERS** table available in your database which you can use to store required information related to customers.

SQL- Drop or Delete Table

The SQL DROP TABLE statement is used to remove a table definition and all data, indexes, triggers, constraints, and permission specifications for that table.

NOTE: You have to be careful while using this command because once a table is deleted then all the information available in the table would also be lost forever.

Syntax:

Basic syntax of DROP TABLE statement is as follows:

```
DROP TABLE table_name;
```

Example:

Let us first verify **CUSTOMERS** table and then we would delete it from the database:

```
SQL> DESC CUSTOMERS;
```

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI		
NAME	varchar(20)	NO			
AGE	int(11)	NO			
ADDRESS	char(25)	YES		NULL	
SALARY	decimal(18,2)	YES		NULL	

5 rows in set (0.00 sec)

This means CUSTOMERS table is available in the database, so let us drop it as follows:

```
SQL> DROP TABLE CUSTOMERS;  
Query OK, 0 rows affected (0.01 sec)
```

Now, if you would try DESC command, then you would get error as follows:

```
SQL> DESC CUSTOMERS;  
ERROR 1146 (42S02): Table 'TEST.CUSTOMERS' doesn't exist
```

SQL –Insert Query

The **SQL INSERT INTO** Statement is used to add new rows of data to a table in the database.

Syntax:

There are two basic syntaxes of **INSERT INTO** statement as follows:

```
INSERT INTO TABLE_NAME (column1, column2, column3, ...columnN)]  
VALUES (value1, value2, value3, ...valueN);
```

Here, **column1, column2,...columnN** are the names of the columns in the table into which you want to insert data.

You may not need to specify the columns name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table. The SQL INSERT INTO syntax would be as follows:

```
INSERT INTO TABLE_NAME VALUES (value1,value2,value3, ...valueN);
```

Example:

Following statements would create six records in CUSTOMERS table:

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );
```

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );
```

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );
```

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );
```

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00 );
```

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (6, 'Komal', 22, 'MP', 4500.00 );
```

You can create a record in CUSTOMERS table using second syntax as follows:

```
INSERT INTO CUSTOMERS
VALUES (7, 'Muffy', 24, 'Indore', 10000.00 );
```

All the above statements would produce the following records in CUSTOMERS table:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Populate one table using another table:

You can populate data into a table through select statement over another table provided another table has a set of fields, which are required to populate first table. Here is the syntax:

```
INSERT INTO first_table_name [(column1, column2, ... columnN)]  
    SELECT column1, column2, ...columnN  
    FROM second_table_name  
    [WHERE condition];
```

SQL- SELECT QUERY

SQL SELECT statement is used to fetch the data from a database table which returns data in the form of result table. These result tables are called result-sets.

Syntax:

The basic syntax of SELECT statement is as follows:

```
SELECT column1, column2, columnN FROM table_name;
```

Here, column1, column2...are the fields of a table whose values you want to fetch.

If you want to fetch **all the fields available** in the table, then you can use the following syntax:

```
SELECT * FROM table_name;
```

Example:

Consider the **CUSTOMERS** table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following is an example, which would fetch **ID**, **Name** and **Salary** fields of the customers available in **CUSTOMERS** table:

```
SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS;
```

This would produce the following result:

ID	NAME	SALARY
1	Ramesh	2000.00
2	Khilan	1500.00
3	kaushik	2000.00
4	Chaitali	6500.00
5	Hardik	8500.00
6	Komal	4500.00
7	Muffy	10000.00

If you want to fetch **all the fields** of CUSTOMERS table, then use the following query:

```
SQL> SELECT * FROM CUSTOMERS;
```

This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

SQL- Where Clause

The **SQL WHERE** clause is used to specify a condition while fetching the data from single table or joining with multiple tables.

If the given condition is satisfied then only it returns specific value from the table.

You would use WHERE clause to filter the records and fetching only necessary records.

The WHERE clause is not only used in SELECT statement, but it is also used in UPDATE, DELETE statement, etc.

Syntax:

The basic syntax of SELECT statement with WHERE clause is as follows:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

You can specify a condition using comparison or logical operators like >, <, =, **LIKE**, **NOT**, etc.

Example:

Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following is an example which would fetch ID, Name and Salary fields from the CUSTOMERS table where salary is **greater than 2000**:

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000;
```

This would produce the following result:

ID	NAME	SALARY
4	Chaitali	6500.00
5	Hardik	8500.00
6	Komal	4500.00
7	Muffy	10000.00

Following is an example, which would fetch ID, Name and Salary fields from the CUSTOMERS table for a customer with name **Hardik**.

Here, it is important to note that all the strings should be given inside single quotes " whereas numeric values should be given without any quote as in above example:

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE NAME = 'Hardik';
```

This would produce the following result:

ID	NAME	SALARY
5	Hardik	8500.00

SQL- AND & OR Conjunctive operators

The SQL AND and OR operators are used to combine multiple conditions to narrow data in an SQL statement. These two operators are called conjunctive operators.

These operators provide a means to make multiple comparisons with different operators in the same SQL statement.

The AND Operator:

The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

Syntax:

The basic syntax of AND operator with WHERE clause is as follows:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] AND [condition2]...AND [conditionN];
```

You can combine **N** number of conditions using AND operator. For an action to be taken by the SQL statement, whether it be a transaction or query, **all conditions separated** by the AND must be **TRUE**.

Example:

Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following is an example, which would fetch ID, Name and Salary fields from the CUSTOMERS table where **salary is greater than 2000 AND age is less than 25 years**:

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000 AND age < 25;
```

This would produce the following result:

ID	NAME	SALARY
6	Komal	4500.00
7	Muffy	10000.00

The OR Operator:

The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

Syntax:

The basic syntax of OR operator with WHERE clause is as follows:

```
SELECT column1, column2, columnN  
FROM table_name  
WHERE [condition1] OR [condition2]...OR [conditionN]
```

You can combine N number of conditions using OR operator. For an action to be taken by the SQL statement, whether it be a transaction or query, **only any ONE** of the conditions separated by the OR must be **TRUE**.

Example:

Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following is an example, which would fetch ID, Name and Salary fields from the CUSTOMERS table where **salary is greater than 2000 OR age is less than 25 years**:

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000 OR age < 25;
```

This would produce the following result:

ID	NAME	SALARY
3	kaushik	2000.00
4	Chaitali	6500.00
5	Hardik	8500.00
6	Komal	4500.00
7	Muffy	10000.00

SQL- Update Query

The **SQL UPDATE** Query is used to modify the existing records in a table.

You can use **WHERE** clause with **UPDATE** query to update selected rows otherwise all the rows would be affected.

Syntax:

The basic syntax of **UPDATE** query with **WHERE** clause is as follows:

```
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
```

You can combine N number of conditions using AND or OR operators.

Example:

Consider the **CUSTOMERS** table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following is an example, which would update **ADDRESS** for a customer whose **ID** is **6**:

```
SQL> UPDATE CUSTOMERS
SET ADDRESS = 'Pune'
WHERE ID = 6;
```

Now, CUSTOMERS table would have the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	Pune	4500.00
7	Muffy	24	Indore	10000.00

If you want to modify all ADDRESS and SALARY column values in CUSTOMERS table, you do not need to use WHERE clause and UPDATE query would be as follows:

```
SQL> UPDATE CUSTOMERS  
SET ADDRESS = 'Pune', SALARY = 1000.00;
```

Now, CUSTOMERS table would have the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Pune	1000.00
2	Khilan	25	Pune	1000.00
3	kaushik	23	Pune	1000.00
4	Chaitali	25	Pune	1000.00
5	Hardik	27	Pune	1000.00
6	Komal	22	Pune	1000.00
7	Muffy	24	Pune	1000.00

SQL- Delete Query

The **SQL DELETE** Query is used to delete the existing records from a table.

You can use **WHERE** clause with DELETE query to delete selected rows, otherwise all the records would be deleted.

Syntax:

The basic syntax of DELETE query with WHERE clause is as follows:

```
DELETE FROM table_name  
WHERE [condition];
```

You can combine N number of conditions using AND or OR operators.

Example:

Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following is an example, which would DELETE a customer, whose ID is 6:

```
SQL> DELETE FROM CUSTOMERS  
WHERE ID = 6;
```

Now, CUSTOMERS table would have the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
7	Muffy	24	Indore	10000.00

If you want to DELETE all the records from CUSTOMERS table, you do not need to use WHERE clause and DELETE query would be as follows:

```
SQL> DELETE FROM CUSTOMERS;
```

Now, CUSTOMERS table would not have any record.

SQL- Like Clause

The **SQL LIKE** clause is used to compare a value to similar values using wildcard operators.

There are two wildcards used in conjunction with the LIKE operator:

1. The percent sign (%)
2. The underscore (_)

The percent sign represents zero, one, or multiple characters. The underscore represents a single number or character. The symbols can be used in combinations.

Syntax:

The basic syntax of % and _ is as follows:

```
SELECT FROM table_name
WHERE column LIKE 'XXXX%'

or

SELECT FROM table_name
WHERE column LIKE '%XXXX%'

or

SELECT FROM table_name
WHERE column LIKE 'XXXX_'

or

SELECT FROM table_name
WHERE column LIKE '_XXXX'

or

SELECT FROM table_name
WHERE column LIKE '_XXXX_'
```

You can combine N number of conditions using AND or OR operators. Here, XXXX could be any numeric or string value.

Example:

Here are number of examples showing WHERE part having different LIKE clause with '%' and '_' operators:

Statement	Description
WHERE SALARY LIKE '200%'	Finds any values that start with 200
WHERE SALARY LIKE '%200%'	Finds any values that have 200 in any position
WHERE SALARY LIKE '_00%'	Finds any values that have 00 in the second and third positions
WHERE SALARY LIKE '2_%_ %'	Finds any values that start with 2 and are at least 3 characters in length
WHERE SALARY LIKE '%2'	Finds any values that end with 2
WHERE SALARY LIKE '_2%3'	Finds any values that have a 2 in the second position and end with a 3
WHERE SALARY LIKE '2___3'	Finds any values in a five-digit number that start with 2 and end with 3

Let us take a real example, consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following is an example, which would display all the records from CUSTOMERS table where SALARY starts with 200:

```
SQL> SELECT * FROM CUSTOMERS  
WHERE SALARY LIKE '200%';
```

This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
3	kaushik	23	Kota	2000.00

SQL- TOP, LIMIT or ROWNUM Clause

The **SQL TOP** clause is used to fetch a TOP N number or X percent records from a table.

Note: All the databases do not support TOP clause. For example MySQL supports LIMIT clause to fetch limited number of records and Oracle uses ROWNUM to fetch limited number of records.

Syntax:

The basic syntax of TOP clause with SELECT statement would be as follows:

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE [condition]
```

Example:

Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following is an example on SQL server, which would fetch top 3 records from CUSTOMERS table:

```
SQL> SELECT TOP 3 * FROM CUSTOMERS;
```

This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00

If you are using MySQL server, then here is an equivalent example:

```
SQL> SELECT * FROM CUSTOMERS  
LIMIT 3;
```

This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00

If you are using Oracle server, then here is an equivalent example:

```
SQL> SELECT * FROM CUSTOMERS  
WHERE ROWNUM <= 3;
```

This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00

SQL- ORDER BY Clause

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. Some database sorts query results in ascending order by default.

Syntax:

The basic syntax of ORDER BY clause is as follows:

```
SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

You can use more than one column in the ORDER BY clause. Make sure whatever column you are using to sort, that column should be in column-list.

Example:

Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following is an example, which would sort the result in ascending order by NAME and SALARY:

```
SQL> SELECT * FROM CUSTOMERS  
      ORDER BY NAME, SALARY;
```

This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
3	kaushik	23	Kota	2000.00
2	Khilan	25	Delhi	1500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00
1	Ramesh	32	Ahmedabad	2000.00

Following is an example, which would sort the result in descending order by NAME:

```
SQL> SELECT * FROM CUSTOMERS  
      ORDER BY NAME DESC;
```

This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
7	Muffy	24	Indore	10000.00
6	Komal	22	MP	4500.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
5	Hardik	27	Bhopal	8500.00
4	Chaitali	25	Mumbai	6500.00

SQL- GROUP BY

The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups.

The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax:

The basic syntax of GROUP BY clause is given below. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2
```

Example:

Consider the CUSTOMERS table is having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

If you want to know the total amount of salary on each customer, then GROUP BY query would be as follows:

```
SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS  
      GROUP BY NAME;
```

This would produce the following result:

NAME	SUM(SALARY)
Chaitali	6500.00
Hardik	8500.00
kaushik	2000.00
Khilan	1500.00
Komal	4500.00
Muffy	10000.00
Ramesh	2000.00

Now, let us have following table where CUSTOMERS table has the following records with duplicate names:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Ramesh	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	kaushik	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Now again, if you want to know the total amount of salary on each customer, then GROUP BY query would be as follows:

```
SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS  
      GROUP BY NAME;
```

This would produce the following result:

NAME	SUM(SALARY)
Hardik	8500.00
kaushik	8500.00
Komal	4500.00
Muffy	10000.00
Ramesh	3500.00

SQL-Distinct Keyword

The SQL **DISTINCT** keyword is used in conjunction with SELECT statement to eliminate all the duplicate records and fetching only unique records.

There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only unique records instead of fetching duplicate records.

Syntax:

The basic syntax of DISTINCT keyword to eliminate duplicate records is as follows:

```
SELECT DISTINCT column1, column2,.....columnN
FROM table_name
WHERE [condition]
```

Example:

Consider the CUSTOMERS table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

First, let us see how the following SELECT query returns duplicate salary records:

```
SQL> SELECT SALARY FROM CUSTOMERS
      ORDER BY SALARY;
```

This would produce the following result where salary 2000 is coming twice which is a duplicate record from the original table.

SALARY
1500.00
2000.00
2000.00
4500.00
6500.00
8500.00
10000.00

Now, let us use DISTINCT keyword with the above SELECT query and see the result:

```
SQL> SELECT DISTINCT SALARY FROM CUSTOMERS  
      ORDER BY SALARY;
```

This would produce the following result where we do not have any duplicate entry:

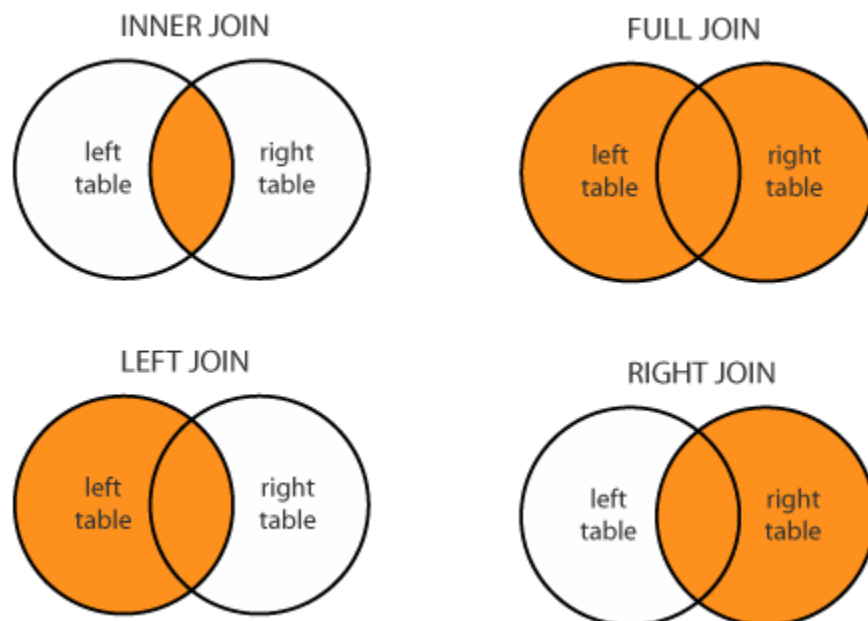
SALARY
1500.00
2000.00
4500.00
6500.00
8500.00
10000.00

SQL- JOIN

The SQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Different types of JOINS

1. **INNER JOIN:** returns rows when there is a match in both tables.
2. **LEFT JOIN:** returns all rows from the left table, even if there are no matches in the right table.
3. **RIGHT JOIN:** returns all rows from the right table, even if there are no matches in the left table.
4. **FULL JOIN:** returns rows when there is a match in one of the tables.
5. **SELF JOIN:** is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
6. **CARTESIAN JOIN:** returns the Cartesian product of the sets of records from the two or more joined tables.



Consider the following two tables, (a) CUSTOMERS table is as follows:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

(b) Another table is ORDERS as follows:

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Now, let us join these two tables in our SELECT statement as follows:

```
SQL> SELECT ID, NAME, AGE, AMOUNT
      FROM CUSTOMERS, ORDERS
      WHERE CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result:

ID	NAME	AGE	AMOUNT
3	kaushik	23	3000
3	kaushik	23	1500
2	Khilan	25	1560
4	Chaitali	25	2060

Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal symbol.

The SQL JOIN syntax

The general syntax is:

```
1.  SELECT column-names
2.    FROM table-name1 JOIN table-name2
3.      ON column-name1 = column-name2
4.    WHERE condition
```

The general syntax with INNER is:

```
1.  SELECT column-names
2.    FROM table-name1 INNER JOIN table-name2
3.      ON column-name1 = column-name2
4.    WHERE condition
```

Note: The INNER keyword is optional: it is the default as well as the most commonly used JOIN operation.

SQL JOIN Examples

ORDER
Id 
OrderDate
OrderNumber
CustomerId
TotalAmount

CUSTOMER
Id 
FirstName
LastName
City
Country
Phone

Problem: List all orders with customer information

```
1.  SELECT OrderNumber, TotalAmount, FirstName, LastName, City, Country
2.    FROM [Order] JOIN Customer
3.      ON [Order].CustomerId = Customer.Id
```

In this example using table **aliases** for [Order] and Customer might have been useful.

Results: 830 records.

OrderNumber	TotalAmount	FirstName	LastName	City	Country
542378	440.00	Paul	Henriot	Reims	France
542379	1863.40	Karin	Josephs	Münster	Germany
542380	1813.00	Mario	Pontes	Rio de Janeiro	Brazil
542381	670.80	Mary	Saveley	Lyon	France
542382	3730.00	Pascale	Cartrain	Charleroi	Belgium
542383	1444.80	Mario	Pontes	Rio de Janeiro	Brazil
542384	625.20	Yang	Wang	Bern	Switzerland

Problem: List all orders with
product names, quantities, and prices

ORDER	ORDERITEM	PRODUCT
Id 	Id 	Id 
OrderDate	OrderId	ProductName
OrderNumber	ProductId	SupplierId
CustomerId	UnitPrice	UnitPrice
TotalAmount	Quantity	Package
		IsDiscontinued

```
1. SELECT O.OrderNumber, CONVERT(date,O.OrderDate) AS Date,  
2.      P.ProductName, I.Quantity, I.UnitPrice  
3. FROM [Order] O  
4. JOIN OrderItem I ON O.Id = I.OrderId  
5. JOIN Product P ON P.Id = I.ProductId  
6. ORDER BY O.OrderNumber
```

This query performs two JOIN operations with 3 tables.

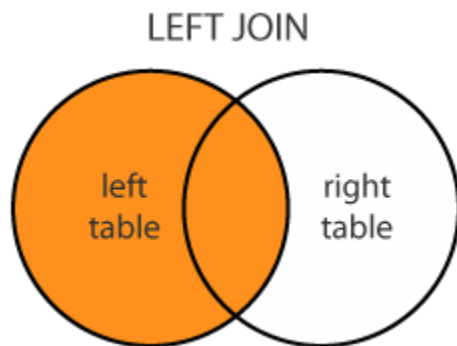
The **O**, **I**, and **P** are table **aliases**. Date is a column alias.

Results: 2155 records

OrderNumber	Date	ProductName	Quantity	UnitPrice
542378	7/4/2012 12:00:00 AM	Queso Cabrales	12	14.00
542378	7/4/2012 12:00:00 AM	Singaporean Hokkien Fried Mee	10	9.80
542378	7/4/2012 12:00:00 AM	Mozzarella di Giovanni	5	34.80
542379	7/5/2012 12:00:00 AM	Tofu	9	18.60
542379	7/5/2012 12:00:00 AM	Manjimup Dried Apples	40	42.40
542380	7/8/2012 12:00:00 AM	Jack's New England Clam Chowder	10	7.70

SQL LEFT JOIN

- **LEFT JOIN** performs a join starting with the first (left-most) table and then any matching second (right-most) table records.
- **LEFT JOIN** and **LEFT OUTER JOIN** are the same.



The SQL LEFT JOIN syntax

The general syntax is:

```
1. SELECT column-names
2. FROM table-name1 LEFT JOIN table-name2
3. ON column-name1 = column-name2
4. WHERE condition
```

The general LEFT OUTER JOIN syntax is:

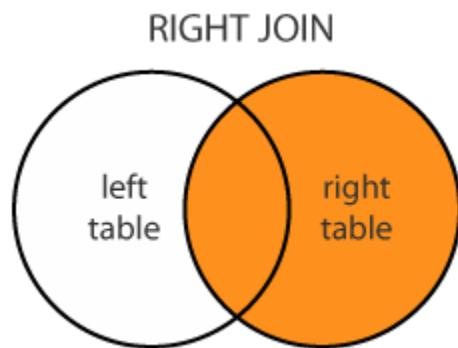
```
1. SELECT OrderNumber, TotalAmount, FirstName, LastName, City, Country
2. FROM Customer C LEFT JOIN [Order] O
3. ON O.CustomerId = C.Id
4. ORDER BY TotalAmount
```

This will list all customers, whether they placed any order or not.

The **ORDER BY TotalAmount** shows the customers without orders first (i.e. TotalAmount is NULL).

SQL RIGHT JOIN

- RIGHT JOIN performs a join starting with the second (right-most) table and then any matching first (left-most) table records.
- RIGHT JOIN and RIGHT OUTER JOIN are the same.



The SQL RIGHT JOIN syntax

The general syntax is:

```
1.  SELECT column-names
2.    FROM table-name1 RIGHT JOIN table-name2
3.      ON column-name1 = column-name2
4.    WHERE condition
```

The general RIGHT OUTER JOIN syntax is:

```
1.  SELECT column-names
2.    FROM table-name1 RIGHT OUTER JOIN table-name2
3.      ON column-name1 = column-name2
4.    WHERE condition
```

SQL RIGHT JOIN Example

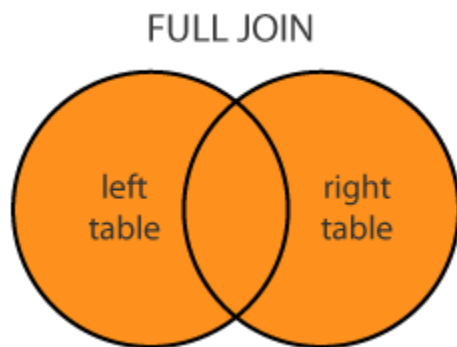
ORDER	CUSTOMER
Id 	Id 
OrderDate	FirstName
OrderNumber	LastName
CustomerId	City
TotalAmount	Country
	Phone

Problem: List customers that have not placed orders

```
1. SELECT TotalAmount, FirstName, LastName, City, Country
2.    FROM [Order] O RIGHT JOIN Customer C
3.        ON O.CustomerId = C.Id
4. WHERE TotalAmount IS NULL
```

SQL FULL JOIN Statement

- FULL JOIN returns all matching records from both tables whether the other table matches or not.
- FULL JOIN can potentially return very large datasets.
- FULL JOIN and FULL OUTER JOIN are the same.



The SQL FULL JOIN syntax

The general syntax is:

```
1.  SELECT column-names
2.    FROM table-name1 FULL JOIN table-name2
3.      ON column-name1 = column-name2
4.  WHERE condition
```

The general FULL OUTER JOIN syntax is:

```
1.  SELECT column-names
2.    FROM table-name1 FULL OUTER JOIN table-name2
3.      ON column-name1 = column-name2
4.  WHERE condition
```

SQL FULL JOIN Examples

Problem: Match all customers and suppliers by country

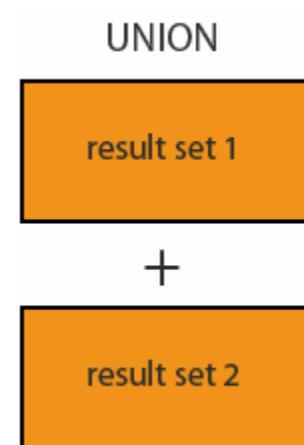
CUSTOMER	SUPPLIER
Id 	Id 
FirstName	CompanyName
LastName	ContactName
City	City
Country	Country
Phone	Phone
	Fax

```
1. SELECT C.FirstName, C.LastName, C.Country AS CustomerCountry,
2.      S.Country AS SupplierCountry, S.CompanyName
3.    FROM Customer C FULL JOIN Supplier S
4.      ON C.Country = S.Country
5.   ORDER BY C.Country, S.Country
```

This returns suppliers that have no customers in their country, and customers that have no suppliers in their country, and customers and suppliers that are from the same country.

SQL UNION Clause

- UNION combines the result sets of two queries.
- Column data types in the two queries must match.
- UNION combines by column position rather than column name.



The SQL UNION syntax

The general syntax is:

```
1.  SELECT column-names
2.    FROM table-name
3.  UNION
4.  SELECT column-names
5.    FROM table-name
```

SQL UNION Examples

Problem: List all contacts, i.e., suppliers and customers

CUSTOMER	SUPPLIER
Id 	Id 
FirstName	CompanyName
LastName	ContactName
City	City
Country	Country
Phone	Phone
	Fax

```
1. SELECT 'Customer' As Type,  
2.     FirstName + ' ' + LastName AS ContactName,  
3.     City, Country, Phone  
4. FROM Customer  
5. UNION  
6. SELECT 'Supplier',  
7.     ContactName, City, Country, Phone  
8. FROM Supplier
```

SQL Sub queries

- A sub query is a SQL query within a query.
- Sub queries are nested queries that provide data to the enclosing query.
- Sub queries can return individual values or a list of records
- Sub queries must be enclosed with parenthesis

The SQL sub query syntax

There is no general syntax; sub queries are regular queries placed inside parenthesis.

Sub queries can be used in different ways and at different locations inside a query:

Here is an sub query with the **IN** operator

```
1.  SELECT column-names
2.    FROM table-name1
3.    WHERE value IN (SELECT column-name
4.                      FROM table-name2
5.                      WHERE condition)
```

Sub queries can also assign column values for each record:

```
1.  SELECT column1 = (SELECT column-name FROM table-name WHERE condition),
2.    column-names
3.    FROM table-name
4.    WEHRE condition
```

SQL Sub query Examples

Problem: List products with order quantities greater than 100

PRODUCT	
Id	
ProductName	
SupplierId	
UnitPrice	
Package	
IsDiscontinued	

ORDERITEM	
Id	
OrderId	
ProductId	
UnitPrice	
Quantity	

```
1. SELECT ProductName
2.   FROM Product
3.  WHERE Id IN (SELECT ProductId
4.                FROM OrderItem
5.                WHERE Quantity > 100)
```